



# Imperceptible QR Watermarks in High-Resolution Videos

Tymoteusz Lindner<sup>1,2</sup><sup>a</sup>, Tomasz Hawro<sup>1</sup> and Piotr Syga<sup>1,3</sup><sup>b</sup>

<sup>1</sup>*Vestigiti, Poland*

<sup>2</sup>*Poznan University of Technology, Poznan, Poland*

<sup>3</sup>*Wroclaw University of Science and Technology, Wroclaw, Poland*

**Keywords:** Watermarking, QR Codes, Encoder-Decoder, Deep Learning.

**Abstract:** The recent advancements in watermarking have indicated the capacity of deep learning for video copyright protection. We introduce a novel deep neural network architecture that uses QR-coded-based messages for video watermarking. Our framework encompasses an encoder-decoder structure, integrating two noiser components, to adeptly increase the robustness against attacks, including MPEG compression. Our solution is aimed at real-life applications; hence we focus on high-resolution videos and intend the encoded image to be indistinguishable from the cover image. To that end, we perform a subjective evaluation on a group of 72 volunteers as well as calculate objective quality metrics obtaining 0.000241 LPIPS, 1.000 SSIM, and 63.8dB PSNR for the best scenario. The obtained results improve PSNR reported by REVMark (Y. Zhang et al., 2023) by around 30dB and LPIPS by a factor of 100. Furthermore, extensive evaluation on both standard COCO dataset and high-resolution videos underlines the method's high robustness against image distortion attacks, achieving over 0.9 bit accuracy for JPEG ( $q=90$ ), Dropout ( $p=0.85$ ) and chroma subsampling (4:2:0).


## 1 INTRODUCTION


In the last few years, there has been growing interest in copyright protection techniques, including watermarking, within the multimedia industry. At the same time, the need to effectively protect intellectual property through widespread copyright marking has become more pressing due to the rising incidence of piracy. Online content distribution platforms present a significant challenge in protecting intellectual property rights due to the ease with which content can be intercepted and redistributed without proper authorization. To address this issue (Giladi, 2017; Hietbrink, 2018), digital watermarking has emerged as a powerful tool for content owners to safeguard their rights. Digital watermarking involves embedding a hidden message within an image or video that can be used to identify the content owner in the event of unauthorized distribution.

One of the primary goals of digital watermarking is to create a robust and transparent marker that can withstand various distortions. At the same time, the watermark must not compromise the quality of the

original content or cause noticeable artifacts for the end-user. To achieve efficient identification of unauthorized distributors, the watermark must contain enough information to uniquely identify each user who accesses the content. However, storing a payload large enough to identify each user while maintaining transparency and minimizing quality loss requires a trade-off. One approach to address these contradictory demands is to use the temporal domain to reduce the payload required in a single frame (Błażkiewicz et al., 2020; Plata & Syga, 2020b). In this paper, we assume that the whole payload has to be stored in each frame, so that the violator may be identified by a single frame.

Another critical factor in digital watermarking is the impact of compression algorithms on watermark effectiveness. To optimize transmission, the images and videos are compressed to remove redundant information that humans cannot perceive, e.g. in JPEG compression, resulting in a loss of detail in the chroma components (Cr and Cb) of the color space. As these components are significantly reduced during compression, watermarks must be embedded into the more visible parts of the image or video, like the

<sup>a</sup> <https://orcid.org/0000-0001-5339-8663>

<sup>b</sup> <https://orcid.org/0000-0002-0266-5802>

luminance (Y channel). Video watermarking can be seen as an extension of image watermarking, as MJPEG compression saves a video as a sequence of JPEG frames. However, other standards like MPEG compression (e.g., HEVC) encode full information using JPEG frames at intervals, while relying on motion vectors to reference previous frames. When evaluating watermarking methods, factors such as transparency, capacity, robustness, and real-time embedding capabilities for video applications are crucial, especially in commercial settings with high-resolution videos.

**Related Work.** The problem of watermarking has been tackled from various angles, including the spatial, temporal, and frequency domains. To enhance robustness, researchers have frequently employed the Discrete Cosine Transform (DCT), which is already used in JPEG compression, wavelets or redundancy. Examples of such approaches can be found in (Hsu & Tu, 2020; Kumar et al., 2020).

Recently, deep learning techniques have gained popularity in watermarking. In this paper (Zhu et al., 2018) authors proposed an end-to-end encoder-noiser-decoder framework, which spreads message embeddings across all pixels. A similar approach using QR codes was evaluated in (P. Zhang et al., 2021), though it pertains to image steganography, which focuses on hiding the message. Note that HiDDeN was not the first watermarking system using machine learning, as extreme machine learning has been used in (Mishra et al., 2012).

A paper (Wen & Aydore, 2019) introduced adversarial training, improving robustness at the cost of transparency. Since the resulting image no longer meets conventional standards of PSNR, it holds significantly limited commercial value and is a purely academic idea. Another research area involves spread spectrum watermarking with adaptive strength and differential quantization, as demonstrated by (Huang et al., 2019) to improve PSNR guarantees. In this paper (Luo et al., 2020) authors studied algorithm robustness by using an additional neural network to generate diverse distortions during training, enhancing accuracy. The authors of (Plata & Syga, 2020a) proposed a method to increase local capacity and robustness against attacks, a topic also explored by (Ahmadi et al., 2018). Mitigating the influence of image compression was the key concern in (Hamamoto & Kawamura, 2020). An important step was made also in (K. A. Zhang et al., 2019), where authors used an attention mechanism. More examples of using machine learning in watermarking can be

found in (Ernawan & Ariatmanto, 2023; Singh & Singh, 2024).

The closest approaches to ours, including the examination of high-resolution videos, were conducted by Chen et al. (Chen et al., 2023), where Zernike moments were utilized. The authors report 47.6dB PSNR, and 0.999 SSIM. However, embedding a watermark in FHD video took almost 0.9s per frame and extraction took 0.3s per frame, making it impractical for real-time use, even for such low resolution. The authors investigate the robustness against various attacks, unfortunately, they use normalized cross-correlation in their estimation instead of the metrics traditionally used (Zhu et al., 2018), hence the results are not directly comparable.

DVMark (Luo et al., 2023) followed by REVMARK (Y. Zhang et al., 2023) focuses on MPEG compression, which is of particular importance in real-life scenarios. The latter reports PSNR of 37.5dB and LPIPS of 0.0296 with accuracy reaching 0.967 for the best possible scenario, with a watermark payload equal to 64bits.

**Contribution.** (1) We propose a new deep neural network for embedding QR code-based video watermarks. (2) We analyze the robustness against typical attacks, achieving up to 0.962 accuracy. (3) We evaluate the quality of the embedded videos objectively and using subjective human evaluation, demonstrating that the watermark is imperceptible with 0.000241 LPIPS, 1.000 SSIM, and 63.8dB PSNR. (4) We evaluate the watermark on a COCO and high-resolution movies, showing that the method is suitable for real-world usage. The watermark may be used to identify copyright violators after just a single frame or, alternatively, used to identify if the required video has been played (using the watermark as a control mechanism for proof of work).

## 2 METHOD

### 2.1 Model Architecture

The goal of the model is to encode an invisible message into an image that can be later recovered with high accuracy. Our watermarking algorithm encodes binary message  $m \in \{0,1\}^L$  into a cover image  $I_c$  with dimensions  $(W, H, C)$ . The binary message is transformed into a QR code using an expanded message generator and then this expanded message  $m_e$  with shape  $(W, H, 1)$  is encoded into a cover image  $I_c$  (Figure 1).

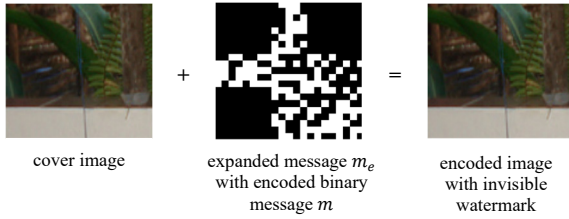


Figure 1: Given a cover image  $I_c$  and expanded message  $m_e$  in the form of a QR code, the model generates an encoded image  $I_e$  with an invisible message; the encoded message can be recovered with high accuracy.

During the recovery process of the message, our model takes an image with an encoded message and returns the recovered binary message  $m_r \in \{0,1\}^L$ .

The proposed architecture consists of five components: watermark generator  $G$ , encoder  $E_\theta$ , decoder  $D_\phi$ , adversarial discriminator  $A_\gamma$ , and a noiser  $N$ , where  $\theta, \phi, \gamma$  are trainable parameters. The architecture is presented in Figure 2.

The watermark generator  $G$  at first generates a binary message  $m$  and then converts it into an expanded message  $m_e$  in the form of a QR code. Expanded message  $m_e$  and a cover image  $I_c$  is used by the encoder  $E_\theta$  to generate an encoded image  $I_e$ :

$$I_e = E_\theta(I_c, m_e) \quad (1)$$

Then the encoded image  $I_e$  is distorted by the noiser  $N$  which applies selected attacks to hinder message

recovery from the encoded image  $I_e$ . Note that some attacks require a cover image  $I_c$ . The output from the noiser  $N$  is the following:

$$I_n = N(I_e, I_c) \quad (2)$$

The decoder  $D_\phi$  recovers the binary message  $m_r$  from the noised image  $I_n$ :

$$m_r = D_\phi(I_n) \in \{0,1\}^L \quad (3)$$

There are two goals of the encoder-decoder architecture: to generate the encoded image  $I_e$ , which is indistinguishable from the cover image  $I_c$ , so that the perceptual difference  $|I_c - I_e|^p$  is close to zero, and to extract a message  $m_r$ , so that the difference  $|m - m_r|$  is zero.

The discriminator  $A_\gamma$  was used for adversarial training to improve visual similarity between encoded and cover images. The adversarial discriminator  $A_\gamma$  distinguishes between fake (encoded) and real (cover) images and outputs the probability  $p_w$  of whether an image is watermarked or not:

$$p_w = A_\gamma(I \in \{I_c, I_e\}) \in [0, 1] \quad (4)$$

During training, we employed the following attacks, which are image processing operations: gaussian blur, gaussian noise, JPEG compression with a quality factor  $q$ , subsampling 4:2:0, dropout, cropping, and cropout. We adopted a double noiser

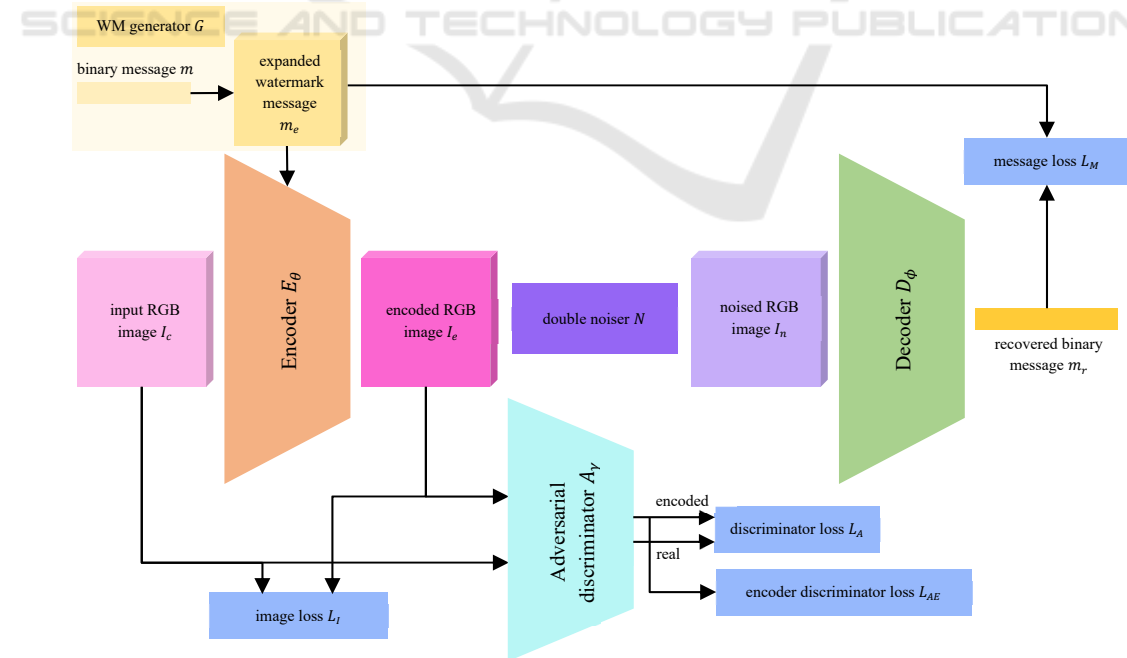


Figure 2: The architecture of the proposed watermarking training pipeline. The watermark generator  $G$  produces an expanded message  $m_e$  which is then embedded into a cover image  $I_c$  by the encoder  $E_\theta$ . The encoded image  $I_e$  is subjected to distortion by the noiser  $N$  to simulate various; the decoder  $D_\phi$  aims to recover the original image  $m_r$ .

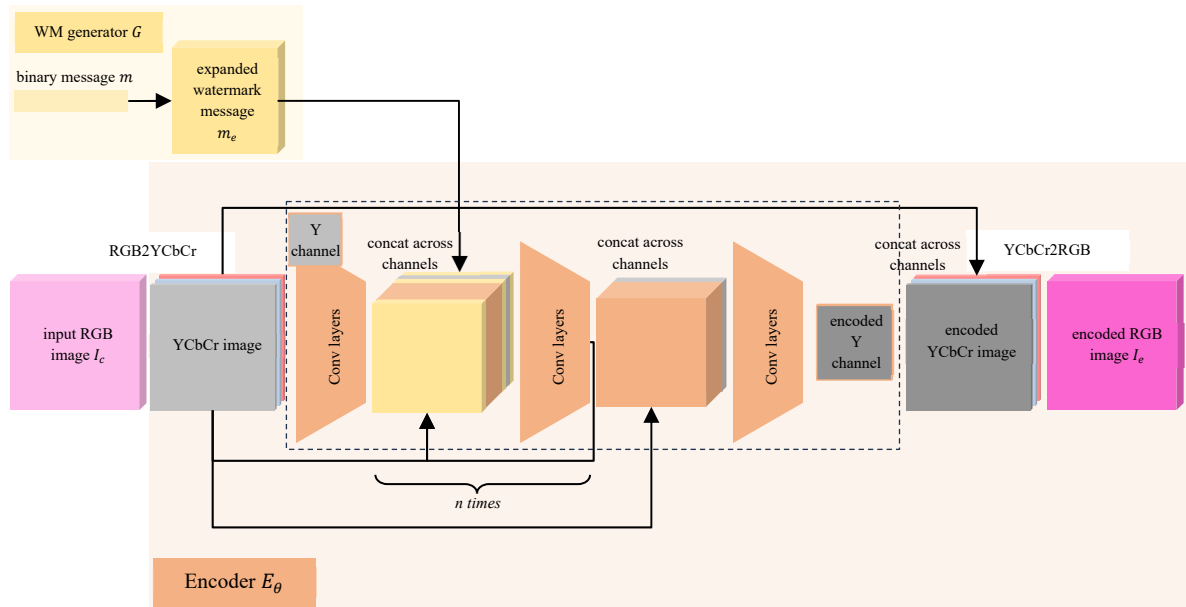


Figure 3: The architecture of the encoder  $E_\theta$  network. The encoder  $E_\theta$  converts the input image  $I_c$  to YCbCr color space. The trainable layers of the encoder marked with the dotted line are fed only by the Y channel of the input image and generate the Y channel for an encoded image. This generated Y channel is then concatenated with Cb and Cr channels from an original input image. At the end encoder  $E_\theta$  converts the encoded YCbCr image to the encoded image  $I_e$  in RGB color space.

approach, where an image was subjected to one of the random attacks mentioned above, and additionally with JPEG compression using a random quality factor, thus input image was always exposed to two different attack types.

The Gaussian blur applies a kernel with a specific size and standard deviation and blurs the image by averaging the color values of neighboring pixels, which reduces sharpness and detail. Gaussian noise adds random noise to the image operating on the pixel level of the image. JPEG compression reduces the file size of an image by discarding some of the image's data. The quality factor  $q$  determines the amount of compression applied, with lower values resulting in more lossy compression and potentially visible artifacts. We used an approximation of the JPEG proposed in (Ahmadi et al., 2018; Plata & Syga, 2020a). Subsampling reduces the resolution of color information in the image, by averaging color values of neighboring pixels. It retains full resolution for the luminance channel but reduces resolution for the chrominance channels. The crop attack returns a cropped square of the encoded image and is parametrized by  $p$ , which specifies the ratio of the squared image to the input image. The cropout attack crops the square of the encoded image and replaces it with the cover image instead of discarding the rest of the image. The cropout attack is also parameterized by  $p$  equal to a ratio of the cropped area over the

entire input image. The dropout attack retains a percentage  $p$  of the pixels in the encoded image, replacing the remaining pixels with their corresponding pixels from the cover image.

## 2.2 Loss Functions

We formulated several loss functions for training our approach  $L_M, L_I, L_{AE}, L_A$ . Each loss function was assigned a weight denoted by  $\lambda$ . The decoder  $D_\phi$  was trained using the message loss function  $L_M$  formulated as:

$$\begin{aligned} L_M &= \lambda_M BCE(m_r, m) \\ &= \frac{1}{L} \sum_L [m \log(m_r) \\ &\quad + (1 - m) \log(1 - m_r)] \end{aligned} \quad (5)$$

thus, decoder loss  $L_D$  was equal to  $L_M$ .

For the encoder  $E_\theta$ , we proposed two combined loss functions. The first one aimed to keep the cover image  $I_c$  and encoded image  $I_e$  as similar as possible, and was defined using the following equation:

$$\begin{aligned} L_I &= \lambda_I MSE(I_c, I_e) + \lambda_S L_{SSIM}(I_c, I_e) \\ &\quad + \lambda_p LPIPS(I_c + I_e) \end{aligned} \quad (6)$$

where  $MSE(I_c, I_e) = \frac{1}{W \cdot H \cdot C} \|I_c - I_e\|_2^2$ . We used three components in the loss function  $L_I$ , the first one is a *Mean Squared Error* function between images  $I_c$  and

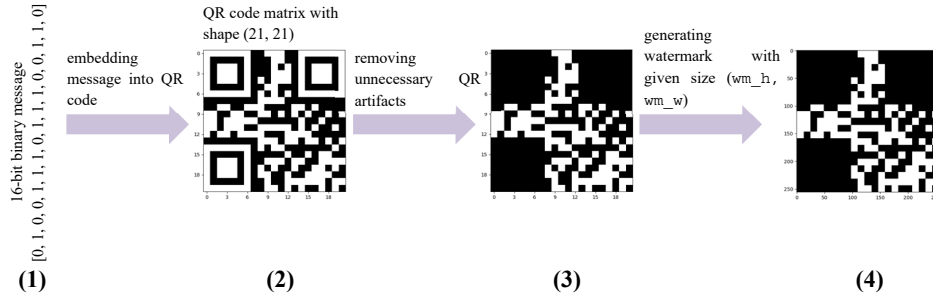


Figure 4: Overview of generating the watermark expanded message. (1) – generating an  $L$ -length binary message using the binary function  $bin(L)$ ; (2) – encoding a binary message into a QR code; (3) – removing artifacts from the QR code matrix (4) – resizing the QR code matrix to desired size  $(W, H)$  using the *nearest* method, resulting in the expanded message  $m_e$ .

$I_e$ , the second one is loss based on *Structural Similarity Index Measure* (SSIM), and the last one is *Learned Perceptual Image Patch Similarity* (LPIPS) (R. Zhang et al., 2018). LPIPS computes the similarity between two image patches using a predetermined neural network, which has been demonstrated to align closely with human perception. In our LPIPS loss function, we used a VGG network with *mean* reduction. Additional tests using Focal MSE loss did not improve the outcome.

Encoder  $E_\theta$  and discriminator  $A_\gamma$  improved of visual similarity of cover  $I_c$  and encoded  $I_e$  images using adversarial training. The goal of the encoder  $E_\theta$  is to generate an image recognized as the cover image by the discriminator  $A_\gamma$ , thus we defined the following loss function:

$$L_{AE} = \lambda_{AE} BCE(A_\gamma(I_e)) = \log(1 - A_\gamma(I_e)) \quad (7)$$

Finally, the loss function  $L_E$  for the encoder  $E_\theta$  was calculated using the formula:

$$L_E = L_I + L_{AE} \quad (8)$$

The goal of the discriminator  $A_\gamma$  was to recognize which image is a cover image  $I_c$  and which one is the encoded image  $I_e$ , thus we formulated the following loss function for the discriminator  $A_\gamma$ :

$$\begin{aligned} L_A &= \lambda_A BCE(A_\gamma(I_e), A_\gamma(I_c)) \\ &= \log(1 - A_\gamma(I_c)) \\ &\quad + \log(A_\gamma(I_e)) \end{aligned} \quad (9)$$

In order to optimize the parameters  $\theta$  and  $\phi$  for the encoder  $E_\theta$  and decoder  $D_\phi$  we performed an optimization algorithm to minimize the following loss function over the distribution of input images and messages:

$$\mathbb{E}_{I_c, m} [L_M(m, m_r) + L_I(I_c, I_e) + L_{AE}(I_e)] \quad (10)$$

Simultaneously the discriminator  $A_\gamma$  was trained to minimize the following objective over the distribution of images:

$$\mathbb{E}_{I_c} [L_A(I_c, I_e)] \quad (11)$$

Note that each loss function  $L_M$ ,  $L_I$ ,  $L_{AE}$ ,  $L_A$  and its components have separate weights  $\lambda$ .

### 2.3 The Architecture of the Networks

The **encoder**  $E_\theta$  model consists of sequential blocks of conv layers with 64 channels, kernel size equal to  $(3, 3)$ , stride, padding equal to  $(1, 1)$ , batch normalization and ReLu activation. Detailed model architecture is presented in Figure 3. The encoder  $E_\theta$  converts the RGB input image  $I_c$  to YCbCr color space. The first three layers of the encoder  $E_\theta$  are standard Conv-Bn-ReLu blocks and are fed only by the Y channel of the input image, thus the encoder  $E_\theta$  has only access to this channel of the image. These three layers generate a feature representation of the input image Y channel. The next two layers, which are also Conv-Bn-ReLu are fed by concatenated expanded message  $m_e$ , feature representation of the input image Y channel and original Y channel of the input image. Note that the expanded message  $m_e$  is used three times in the concatenation list. The final layer is fed by the output from the previous layer concatenated once again with the original Y channel of the input image, this final layer generates a Y channel which is concatenated with channels Cb and Cr from the original image creating an encoded image in the YCbCr color space. At the very end, encoder  $E_\theta$  converts the encoded YCbCr image to the encoded image  $I_e$  in RGB color space. The encoder  $E_\theta$  has a fairly simple architecture with a small number of parameters. It is crucial in the context of the time efficiency of the model because the encoder needs to process data in real-time.



The **decoder**  $D_\phi$  takes the noised image  $I_n$  as an input and converts it to the YCbCr color space. The converted image is fed into the ResNet18 backbone. The next layer in the decoder model is adaptive averaging pooling. Then there were two linear layers with 64 neurons separated with batch normalization and finalized with sigmoid activation.

The adversarial **discriminator**  $A_\gamma$  consists of three sequential blocks of conv layers with 64 channels, kernels equal to (3, 3), stride and padding equal to (1, 1), batch normalization and ReLu activation. These three blocks are followed by an adaptive averaging pooling, one linear layer with 64 neurons and sigmoid activation. The discriminator  $A_\gamma$  returns the probability of whether the input image is a fake (encoded) or real (cover) image.

## 2.4 Watermark Generator

Generating the expanded message  $m$ , which served as the input to the encoder  $E_\theta$ , involved several steps. The individual steps of generating the message  $m_e$  are presented in Figure 4. Initially, we generated a random binary message  $m$  using the binary function  $bin(L)$ , which takes the message length  $L$  as input. In the subsequent step, the binary message  $m$  was converted to a string and then encoded into a QR code matrix. We applied a high error correction level, which can recover 30% of the input data. The mask for the QR code was automatically chosen based on the input data. The generated QR code was (21, 21) pixels.

A QR code matrix contains artifacts for positioning, orientation, alignment, version, format, etc., that are unnecessary because they do not contain data information. During the message creation process, all artifacts of the QR code matrix are removed. The last step in the process of message generation is extrapolating the QR code matrix to the desired size  $(W, H)$  using the *nearest* method, ensuring that we obtain an expanded message  $m_e$ , that has the same width and height size as the input cover image  $I_c$ .

## 2.5 Training Details

We trained our model on two datasets. The first was a subset of the COCO dataset (Lin et al., 2015) consisting of 10000 images for training and 1000 for validation. The second dataset was proprietary, containing frames from diverse video content such as movie trailers, ads, football matches, podcasts, YouTube videos etc. Our custom dataset comprised 228252 images for training and 57073 for validation.

During the research phase, we primarily used the COCO subset for its size, facilitating rapid testing and exploration of various approaches.

The loss weights parameters  $\lambda_M$ ,  $\lambda_I$ ,  $\lambda_S$ ,  $\lambda_P$ ,  $\lambda_{AE}$ , and  $\lambda_A$  equal to 0.5, 0.05, 0.2, 2.0, 0.1, and 1.0 respectively.

During training, we randomly cropped images from the dataset to a size of (128, 128) pixels, resulting cover image  $I_c$ . The expanded message  $m_e$  had the same size as the cover image  $I_c$  (128, 128). This procedure simulated embedding the watermark on a selected area of the entire image frame. A binary message with a length  $L = 16$  was encoded.

To achieve the appropriate balance between high message decoding accuracy and the minimal perceptual difference between the cover  $I_c$  and the encoded  $I_e$  images, we employed several tricks. The first involved pretraining decoder  $D_\phi$  to develop a "general" understanding of retrieving the binary message  $m$  from the expanded message  $m_e$ . During pretraining, we fed the decoder  $D_\phi$  with mixed images combining cover images  $I_c$  and expanded messages  $m_e$  with random proportions ranging from 0 to 1. This resulted in decoder  $D_\phi$  inputs with images where the expanded message  $m_e$  could be barely or clearly visible. Additionally, each image during decoder pretraining was noised with JPEG compression and Gaussian blur.

We used Stochastic Gradient Descent (SGD) with an initial learning rate  $\alpha = 0.001$  for all three trainable modules: the encoder  $E_\theta$ , decoder  $D_\phi$ , and discriminator  $A_\gamma$ , along with a momentum of 0.9 and Nesterov acceleration. During training, we employed schedulers for both learning rates  $\alpha$  and loss weights  $\lambda$  for the encoder  $E_\theta$  and decoder  $D_\phi$ . The loss weight scheduler was activated after 50% of training epochs, where the loss weights for the encoder  $E_\theta$  were increased and for the decoder were decreased. Detailed factors for each loss weight were presented in equation 12 for the encoder  $E_\theta$  and in equation 13 for the decoder  $D_\phi$ . The notation  $\lambda_x^{e=0}$  represents loss weight in the epoch  $e = 0$  for the loss function  $x$ .

$$\lambda_{I,S,P,AE} = \begin{cases} \lambda_{I,S,P,AE}^{e=0} \cdot 1.0 & \text{if } \leq 50\% \text{ train epochs} \\ \lambda_{I,S,P,AE}^{e=0} \cdot 1.5 & \text{if } 50\% < \text{train epochs} \leq 80\% \\ \lambda_{I,S,P,AE}^{e=0} \cdot 2.0 & \text{if } > 80\% \text{ train epochs} \end{cases} \quad (12)$$

$$\lambda_M = \begin{cases} \lambda_M^{e=0} \cdot 1.0 & \text{if } \leq 50\% \text{ train epochs} \\ \lambda_M^{e=0} \cdot 0.5 & \text{if } 50\% < \text{train epochs} \leq 80\% \\ \lambda_M^{e=0} \cdot 0.25 & \text{if } > 80\% \text{ train epochs} \end{cases} \quad (13)$$

The learning rate scheduler was also activated after 50% of the training epochs, and the learning rates decreased for both the encoder  $E_\theta$  and the decoder  $D_\phi$ . After 50% of the epochs, the learning rates were 0.5 times smaller, after 70% 0.1 times smaller, and after 90% 0.001 times smaller.

During the training of the entire pipeline, including the encoder  $E_\theta$ , decoder  $D_\phi$  and discriminator  $A_\gamma$ , we observed that the encoder  $E_\theta$  struggled to invisibly encode watermark on dimmed and dark images. To address this issue, we used an image transformation that adjusted the brightness of the input images. Each input image had a 5% chance of undergoing brightness correction in the range from 0 to 0.3, where 0 represents a completely dark image and 1 represents the original image. The model was trained with a batch size of 64 for 200 epochs.

### 3 EXPERIMENTS

We evaluated two models, one trained on a subset of the COCO dataset and the second one trained on our custom dataset. Both models were assessed on the validation COCO dataset, comprising 1000 images. Additionally, we evaluated the model trained on our custom dataset on 315 short videos that lasted from several seconds to several minutes. During the evaluation, we applied the watermark only to a fixed, small portion of the image, which served as the cover image  $I_c$ . The high-resolution input image size was (784, 784) for evaluation on the COCO dataset and (1920, 1080) for evaluation on short videos, while the watermark size was (128, 128).

To further enhance the invisibility of the watermark, albeit at the expense of decreasing message recovery accuracy, we implemented a technique we called *Gaussian smoothing* (GS). This technique is only applied during inference and can be easily disabled. This approach helped reduce the visibility of the watermark in an entire high-resolution input image. GS involved generating a mask using Gaussian blur with kernel size and standard deviation equal to the watermark size, which was 128. This mask was used to merge the cover  $I_c$  and the encoded  $I_e$  images with the watermark, using the proportion of each pixel value in the mask. In the resulting image after applying GS, brighter areas had a higher proportion of pixel values from the encoded image  $I_e$ , while darker areas had more pixel values from the cover image  $I_c$ . As a result, at the edges, the image with the watermark was much more similar to neighboring pixels from the entire image.

**Used Metrics.** For evaluating message recovery, we used two metrics: bit accuracy (BA) and message accuracy (MA). The BA assesses the fidelity of binary message recovery by comparing recovered messages  $m_r$  with the original binary messages  $m$ , calculating the proportion of correctly matched bits. The MA evaluates the accuracy of message recovery by comparing recovered messages  $m_r$  with the original binary messages  $m$ , assessing whether all bits in each message match. To assess the visual similarity between cover  $I_c$  and encoded  $I_e$  images we used LPIPS, SSIM, and PSNR.

In the following subsections, we present results in tables containing metric values for particular types of attacks. Each table header displays the name of the metrics with an arrow indicating whether higher or lower values of the metric are more desirable. Each column has its color scale, representing the best values for each attack. The color scale transitions smoothly between three colors: green (best), orange, and red (worst).

#### 3.1 Evaluation on the COCO Dataset

This section presents the results of the evaluation of two models on the validation subset of the COCO dataset. Additionally, we tested how GS influences the robustness of our model and improves watermark transparency.

During the evaluation, a different random binary message  $m$  was generated for each image, and it was embedded only in that particular image - each image had a watermark embedded with a different message  $m$ . Consequently, the decoder had only one image (chance) to correctly decode the message. The fragment of the image with size (128, 128) - the cover image  $I_c$  on which the watermark was embedded and which served as the input to the encoder  $E_\theta$ , was always located in the center of the high-resolution input image with size (784, 784). The time taken to embed the watermark in one frame of the image was below 5ms, while the message decoding time was below 3ms.

The model trained on the COCO dataset (Table 1) achieved very similar visual metrics for all types of attacks. The average LPIPS was 0.0014, the SSIM was close to 1.0, and the average PSNR was 49.6dB. The best values for SSIM and LPIPS occurred when the Gaussian blur attack was applied, and for PSNR when no attack was applied. The worst visual metrics were observed for JPEG attacks, but the difference between the best and worst metrics was 0.22 dB for PSNR and below  $1 \cdot 10^{-4}$  for SSIM and LPIPS.

The highest accuracy in decoding a message was observed when no attack was applied and for JPEG compression with  $q=90$ . For both attacks, the BA was above 0.9, while the MA was approximately 0.25. The worst results were observed for crop attacks, where for cropping factors from 0.8 to 0.9, the BA

was 0.5, resulting in the recovery of a completely random message. Only for a cropping factor of 0.95, the BA was 0.55, which still yielded a very poor result. For all crop attacks, the decoder  $D_\phi$  was unable to correctly recover the entire message  $m$  from even a single image, resulting in an MA of 0.0.

Table 1: Results of the evaluation on the COCO dataset for the model trained on the COCO dataset without GS.

Attack name	Bit accuracy $\uparrow$	Message accuracy $\uparrow$	LPIPS $\downarrow$	SSIM $\uparrow$	PSNR $\uparrow$
Identity	0.901	0.243	0.00140	0.999	49.7
JPEG( $q=30$ )	0.846	0.086	0.00144	0.999	49.5
JPEG( $q=50$ )	0.884	0.169	0.00144	0.999	49.5
JPEG( $q=70$ )	0.900	0.222	0.00144	0.999	49.5
JPEG ( $q=90$ )	0.914	0.266	0.00144	0.999	49.5
Gaussian blur ( $\sigma=(0.5, 4)$ )	0.892	0.191	0.00138	0.999	49.6
Gaussian noise ( $\sigma=0.04$ )	0.881	0.137	0.00139	0.999	49.6
Dropout ( $p=0.85$ )	0.864	0.099	0.00139	0.999	49.6
Subsampling (4:2:0)	0.901	0.196	0.00139	0.999	49.6
Crop ( $p=0.8$ )	0.507	0.000	0.00140	0.999	49.7
Crop ( $p=0.85$ )	0.500	0.000	0.00139	0.999	49.7
Crop ( $p=0.9$ )	0.497	0.000	0.00139	0.999	49.7
Crop ( $p=0.95$ )	0.546	0.000	0.00139	0.999	49.7
Cropout ( $p=0.8$ )	0.713	0.024	0.00140	0.999	49.7
Cropout ( $p=0.85$ )	0.769	0.036	0.00139	0.999	49.7
Cropout ( $p=0.9$ )	0.826	0.074	0.00140	0.999	49.7
Cropout ( $p=0.95$ )	0.879	0.166	0.00139	0.999	49.7

Table 2: Results of the evaluation on the COCO dataset for the model trained on our custom dataset without GS.

Attack name	Bit accuracy $\uparrow$	Message accuracy $\uparrow$	LPIPS $\downarrow$	SSIM $\uparrow$	PSNR $\uparrow$
Identity	0.948	0.435	0.000997	0.999	54.3
JPEG( $q=30$ )	0.815	0.081	0.000992	0.999	54.1
JPEG( $q=50$ )	0.902	0.227	0.000992	0.999	54.1
JPEG( $q=70$ )	0.934	0.360	0.000992	0.999	54.1
JPEG ( $q=90$ )	0.959	0.547	0.000992	0.999	54.1
Gaussian blur ( $\sigma=(0.5, 4)$ )	0.930	0.326	0.000933	1.000	54.4
Gaussian noise ( $\sigma=0.04$ )	0.856	0.094	0.001127	0.999	54.1
Dropout ( $p=0.85$ )	0.920	0.285	0.001127	0.999	54.1
Subsampling (4:2:0)	0.947	0.438	0.001127	0.999	54.1
Crop ( $p=0.8$ )	0.505	0.000	0.000996	0.999	54.3
Crop ( $p=0.85$ )	0.501	0.000	0.001008	0.999	54.3
Crop ( $p=0.9$ )	0.506	0.000	0.000998	0.999	54.3
Crop ( $p=0.95$ )	0.568	0.000	0.000995	0.999	54.3
Cropout ( $p=0.8$ )	0.749	0.040	0.001000	0.999	54.3
Cropout ( $p=0.85$ )	0.793	0.050	0.000998	0.999	54.3
Cropout ( $p=0.9$ )	0.860	0.108	0.000997	0.999	54.3
Cropout ( $p=0.95$ )	0.902	0.210	0.000989	0.999	54.3



Table 3: Results of the evaluation on the COCO dataset for the model trained on our custom dataset with GS.

Attack name	Bit accuracy↑	Message accuracy↑	LPIPS↓	SSIM↑	PSNR↑
Identity	0.911	0.250	0.000592	1.000	55.8
JPEG( $q=30$ )	0.753	0.023	0.000620	1.000	55.6
JPEG( $q=50$ )	0.851	0.111	0.000620	1.000	55.6
JPEG( $q=70$ )	0.891	0.168	0.000620	1.000	55.6
JPEG ( $q=90$ )	0.927	0.309	0.000620	1.000	55.6
Gaussian blur ( $\sigma=(0.5, 4)$ )	0.871	0.133	0.000593	1.000	55.8
Gaussian noise ( $\sigma=0.04$ )	0.801	0.033	0.000608	1.000	55.8
Dropout ( $p=0.85$ )	0.871	0.131	0.000608	1.000	55.8
Subsampling (4:2:0)	0.910	0.243	0.000608	1.000	55.8
Crop ( $p=0.8$ )	0.505	0.000	0.000597	1.000	55.8
Crop ( $p=0.85$ )	0.501	0.000	0.000600	1.000	55.8
Crop ( $p=0.9$ )	0.506	0.000	0.000596	1.000	55.8
Crop ( $p=0.95$ )	0.559	0.000	0.000596	1.000	55.8
Cropout ( $p=0.8$ )	0.738	0.033	0.000600	1.000	55.8
Cropout ( $p=0.85$ )	0.781	0.036	0.000600	1.000	55.8
Cropout ( $p=0.9$ )	0.842	0.070	0.000596	1.000	55.8
Cropout ( $p=0.95$ )	0.870	0.125	0.000596	1.000	55.8

The model trained on our custom dataset (Table 2) achieved better visual and accuracy metrics when tested on the validation subset sampled from the COCO dataset. The average LPIPS was 0.001, the SSIM was close to 1.0, and the average PSNR was 54.2dB, representing a 38% improvement for LPIPS and a 9% improvement for PSNR compared to the model trained on the COCO dataset (the improvement in SSIM was negligibly small). The best values again occurred when the Gaussian blur attack was applied, while the worst values were observed for Gaussian noise, dropout, and subsampling. The difference between the best and worst metric values was 0.3dB for PSNR and below  $1 \cdot 10^{-4}$  for SSIM and LPIPS.

Comparing the message recovery metrics between these two models trained on different datasets, improvements can also be noticed. Once again, the highest accuracy in decoding a message was observed for attacks identity and JPEG compression with a quality factor of  $q=90$ . For both attacks, the BA was close to 0.95, while the MA was approximately 0.5. The most significant improvement compared to the model trained on the COCO dataset can be noticed for MA, which was three times better for dropout and two times better for identity, JPEG( $q=90$ ), and subsampling attacks. For the rest of the attacks, MA improved in the range between 30% and 70%. Only for two attacks, MA decreased, for JPEG( $q=30$ ) by

6% and for Gaussian noise by 31%. The BA showed an improvement in the range from 2% to 7%, except for attacks JPEG( $q=30$ ) and Gaussian noise, where BA decreased by approximately 3%. The worst results were observed again for crop attacks.

During the evaluation with GS, only the model trained on our custom dataset was tested, as it yielded better results during the initial evaluation stage. The results of tests with GS for the model trained on our custom dataset are presented in Table 3. For different types of attacks, we observed LPIPS improvement in the range from 57% up to 86%, and PSNR improvement in the range from 2.5% to 3.2% compared to the model without GS. The improvement in visual metrics came at the expense of message recovery metrics. The BA decreased from 1.4% to 7.6% for particular attacks, and the MA from 18% to 72%. The biggest drop was observed for JPEG compression with a quality factor  $q=30$ . For the model trained on our custom dataset without GS applied, the BA was above 0.85 for 10 attacks, whereas with GS, it was above 0.85 for 8 attacks.

We compared the images without encoded watermarks with those with encoded watermarks in the middle of the high-resolution, input image. The examples of the encoded images are presented in Figure 5.



Figure 5: Examples of input image from the COCO dataset without watermark (first row); images with encoded watermark in the middle of the image (second row); the min-max difference between images with and without watermark (third row).

### 3.2 Evaluation on Videos

Embedding watermarks in videos simulates real-world scenarios of applying content watermarking. This section presents the results of evaluating the model trained on our custom dataset with and without GS applied. The videos had a resolution of (1920, 1080), and we encoded the watermark in the middle of every frame of the videos. As with the tests on the COCO dataset, the time taken to embed the watermark in one frame of the image was below 5ms, while the message decoding time was below 3ms. The results from this evaluation are presented in Table 4 and Table 5.

For each video, a random binary message  $m$  was generated, and then we encoded the watermark with this message in the middle of every frame of the video. For every video, the decoder had to decode the same message from each frame. We calculated BA, MA, LPIPS, SSIM, and PSNR for every frame in each video. The final metrics were averaged across all metrics from particular frames. We also calculated additional statistics for every movie. We averaged all recovered messages from particular frames to calculate the final video recovered message from all frames of the video. Based on that, we were able to

calculate the number of correct and incorrect bits for every video and the percentage of correct bits.

The model trained on our custom dataset without GS (Table 4) showed similar performance during the evaluation on videos compared to the evaluation on the COCO dataset. The LPIPS was two times better, and PSNR had a 14% improvement compared to the evaluation on the COCO dataset. When it comes to message recovery metrics, namely BA and MA, we observed the worst results only for JPEG compression and Gaussian noise attacks. The number of incorrect bits was higher than 1 bit only for JPEG compression with quality factors  $q=50$  and  $q=30$  and Gaussian noise attacks. For identity and subsampling attacks, our approach was able to recover all bits for 99.5% of the videos. For 7 out of 9 attacks, we were able to correctly recover all bits for 90% of the tested videos.

The model tested with GS (Table 5) showed a similar performance pattern to the one without GS. The LPIPS was over two times better, and PSNR had a 14% improvement, achieving over 63dB compared to the evaluation on the COCO dataset. The BA and MA decreased only for JPEG compression and Gaussian noise attacks. Similar to the model tested without GS, the approach with GS was able to correctly recover all bits for over 90% of the videos for 7 out of 9 attacks.

Table 4: Results of the evaluation on videos for the model trained on our custom dataset without GS.

Attack name	Bit accuracy $\uparrow$	Message accuracy $\uparrow$	LPIPS $\downarrow$	SSIM $\uparrow$	PSNR $\uparrow$	% correct bits $\uparrow$	# correct bits $\uparrow$	# incorrect bits $\downarrow$
Identity	0.962	0.572	0.000560	1.000	61.8	0.995	15.9	0.1
JPEG( $q=30$ )	0.699	0.011	0.000526	1.000	61.9	0.817	13.1	2.9
JPEG( $q=50$ )	0.799	0.058	0.000526	1.000	61.9	0.916	14.6	1.4
JPEG( $q=70$ )	0.875	0.156	0.000526	1.000	61.9	0.970	15.5	0.5
JPEG ( $q=90$ )	0.947	0.449	0.000526	1.000	61.9	0.994	15.9	0.1
Gaussian blur ( $\sigma=(0.5, 4)$ )	0.938	0.392	0.000518	1.000	62.0	0.993	15.9	0.1
Gaussian noise ( $\sigma=0.04$ )	0.733	0.008	0.000648	1.000	61.5	0.740	11.8	4.2
Dropout ( $p=0.85$ )	0.937	0.366	0.000649	1.000	61.5	0.991	15.9	0.1
Subsampling (4:2:0)	0.962	0.555	0.000649	1.000	61.5	0.995	15.9	0.1

Table 5: Results of the evaluation on videos for the model trained on our custom dataset with GS.

Attack name	Bit accuracy $\uparrow$	Message accuracy $\uparrow$	LPIPS $\downarrow$	SSIM $\uparrow$	PSNR $\uparrow$	% correct bits $\uparrow$	# correct bits $\uparrow$	# incorrect bits $\downarrow$
Identity	0.935	0.375	0.000252	1.000	63.6	0.993	15.9	0.1
JPEG( $q=30$ )	0.665	0.005	0.000241	1.000	63.8	0.819	13.1	2.9
JPEG( $q=50$ )	0.759	0.027	0.000241	1.000	63.8	0.924	14.8	1.2
JPEG( $q=70$ )	0.828	0.071	0.000241	1.000	63.8	0.967	15.5	0.5
JPEG ( $q=90$ )	0.910	0.251	0.000241	1.000	63.8	0.989	15.8	0.2
Gaussian blur ( $\sigma=(0.5, 4)$ )	0.886	0.165	0.000247	1.000	63.6	0.980	15.7	0.3
Gaussian noise ( $\sigma=0.04$ )	0.662	0.003	0.000265	1.000	63.6	0.698	11.2	4.8
Dropout ( $p=0.85$ )	0.901	0.213	0.000265	1.000	63.6	0.982	15.7	0.3
Subsampling (4:2:0)	0.933	0.357	0.000265	1.000	63.6	0.990	15.8	0.2

Comparing our results to other approaches, we achieved a significant improvement in the invisibility of the watermark while maintaining a similar level of message recovery accuracy. During the evaluation of our model in a real-world environment, we obtained a BA of 0.962, LPIPS of 0.00056, SSIM of 1.0, and PSNR of 61.8dB for the scenario where the encoded image was not subjected to any image distortion. Furthermore, we even further improved the invisibility of the watermark, achieving a slightly lower BA of 0.935, LPIPS of 0.000252, SSIM of 1.0, and PSNR of 63.6dB for the scenario where the encoded image was not attacked. The authors of (Chen et al., 2023) reported a PSNR of 47.6dB and SSIM of 0.999, which is approximately 30% worse PSNR than in our approach. The authors of the DVMark model (Luo et al., 2023) achieved slightly higher accuracy in message decoding, with a reported accuracy of 0.967, but they reported almost two times worse PSNR (37.5dB) and over 100x worse LPIPS (0.0296).

### 3.3 Subjective Perceptual Evaluation

Despite using objective quality metrics like LPIPS, SSIM, and PSNR, evaluating the visual experience of end users is crucial for a transparent watermarking system, as individual perceptual experiences vary. To estimate subjective perceptual quality, we performed a MOS evaluation with 72 volunteers aged 23 to 55, watching videos on their own devices. All had a technical background in computer science but not necessarily in image processing. They were informed about the research purpose and performed two tasks. First, they rated the quality of five 5-second video fragments on a scale from 1 (lowest) to 5 (highest). The videos included unmodified, QR-watermarked, and QR-watermarked with GS. The average ratings were 3.48 (std 0.51) for unmodified, 3.39 (std 0.47) for QR-watermarked and 3.47 (std 0.31) for QR-watermarked with GS. After testing with Shapiro-Wilk and performing ANOVA we obtained  $f < 0.013$ , hence there is no significant difference between the ratings of all three groups.

Next, the volunteers were presented with pairs of random frames, one unmodified and one watermarked (with or without GS), and asked to identify the modified frame. This task aimed to ascertain the transparency of the watermark when a reference frame is provided. Each user received 4 random frame pairs. The watermark was correctly identified in 62.5% of cases without GS and 56.25% with GS. These results indicate that the distinction is not significantly higher than random guessing.

## 4 CONCLUSIONS

In the paper, we presented a novel QR-based watermarking approach suitable for real-world applications, allowing messages to be embedded in each frame. We demonstrated two modes of our watermark: a robust mode (without *Gaussian smoothing*) and a transparent mode (with *Gaussian smoothing*). The quality results were significantly improved, with LPIPS reduced by a factor of 100 and PSNR increased by 30dB. This method also features faster embedding, enabling real-time application. Additionally, tests with volunteers showed that the watermarked materials were indistinguishable from the originals. For future work, we aim to enhance watermark localization in cases of cropping and evaluate robustness against a broader spectrum of attacks while maintaining quality, capacity, and embedding time.

## ACKNOWLEDGEMENTS

The research was partially supported by grant number POIR.01.01.01-00-0090/22.

## REFERENCES

- Ahmadi, M., Norouzi, A., Soroushmehr, S. M. R., Karimi, N., Najarian, K., Samavi, S., & Emami, A. (2018). *ReDMark: Framework for Residual Diffusion Watermarking on Deep Networks* (arXiv:1810.07248). arXiv. <http://arxiv.org/abs/1810.07248>
- Błażkiewicz, P., Klonowski, M., & Syga, P. (2020). Droppix: Towards More Realistic Video Fingerprinting. *Proceedings of the 17th International Joint Conference on E-Business and Telecommunications - SECRIPT*, 468–476. <https://doi.org/10.5220/0009876104680476>
- Chen, S., Malik, A., Zhang, X., Feng, G., & Wu, H. (2023). A Fast Method for Robust Video Watermarking Based on Zernike Moments. *IEEE Transactions on Circuits and Systems for Video Technology*, 33(12), 7342–7353. <https://doi.org/10.1109/TCSVT.2023.3281618>
- Ernawan, F., & Ariatmanto, D. (2023). A recent survey on image watermarking using scaling factor techniques for copyright protection. *Multimedia Tools and Applications*, 82(18), 27123–27163. <https://doi.org/10.1007/s11042-023-14447-5>
- Giladi, A. (2017, October 26). *Integrating forensic watermarking into adaptive streaming workflow*. IBC.
- Hamamoto, I., & Kawamura, M. (2020). Neural Watermarking Method Including an Attack Simulator against Rotation and Compression Attacks. *IEICE Transactions on Information and Systems*, E103.D(1), 33–41. <https://doi.org/10.1587/transinf.2019MUP0007>
- Hietbrink, E. (2018). *Forensic Watermarking Implementation Considerations for Streaming Media Created and Approved by the Streaming Video Alliance July 19 , 2018*.
- Hsu, C.-S., & Tu, S.-F. (2020). Enhancing the robustness of image watermarking against cropping attacks with dual watermarks. *Multimedia Tools and Applications*, 79(17), 11297–11323. <https://doi.org/10.1007/s11042-019-08367-6>
- Huang, Y., Niu, B., Guan, H., & Zhang, S. (2019). Enhancing Image Watermarking With Adaptive Embedding Parameter and PSNR Guarantee. *IEEE Transactions on Multimedia*, 21(10), 2447–2460. <https://doi.org/10.1109/TMM.2019.2907475>
- Kumar, C., Singh, A. K., & Kumar, P. (2020). Improved wavelet-based image watermarking through SPIHT. *Multimedia Tools and Applications*, 79(15), 11069–11082. <https://doi.org/10.1007/s11042-018-6177-0>
- Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., & Dollár, P. (2015). *Microsoft COCO: Common Objects in Context* (arXiv:1405.0312). arXiv. <https://doi.org/10.48550/arXiv.1405.0312>
- Luo, X., Li, Y., Chang, H., Liu, C., Milanfar, P., & Yang, F. (2023). DVMark: A Deep Multiscale Framework for Video Watermarking. *IEEE Transactions on Image Processing*, 1–1. <https://doi.org/10.1109/TIP.2023.3251737>
- Luo, X., Zhan, R., Chang, H., Yang, F., & Milanfar, P. (2020). Distortion Agnostic Deep Watermarking. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 13545–13554. <https://doi.org/10.1109/CVPR42600.2020.01356>
- Mishra, A., Goel, A., Singh, R., Chetty, G., & Singh, L. (2012). A novel image watermarking scheme using Extreme Learning Machine. *The 2012 International Joint Conference on Neural Networks (IJCNN)*, 1–6. <https://doi.org/10.1109/IJCNN.2012.6252363>
- Plata, M., & Syga, P. (2020a). Robust Spatial-spread Deep Neural Image Watermarking. *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 62–70. <https://doi.org/10.1109/TrustCom50675.2020.00022>



- Plata, M., & Syga, P. (2020b). *Robust watermarking with double detector-discriminator approach* (arXiv:2006.03921). arXiv. <https://doi.org/10.48550/arXiv.2006.03921>
- Singh, H. K., & Singh, A. K. (2024). Digital image watermarking using deep learning. *Multimedia Tools and Applications*, 83(1), 2979–2994. <https://doi.org/10.1007/s11042-023-15750-x>
- Wen, B., & Aydore, S. (2019). *ROMark: A Robust Watermarking System Using Adversarial Training* (arXiv:1910.01221). arXiv. <https://doi.org/10.48550/arXiv.1910.01221>
- Zhang, K. A., Xu, L., Cuesta-Infante, A., & Veeramachaneni, K. (2019). *Robust Invisible Video Watermarking with Attention* (arXiv:1909.01285). arXiv. <https://doi.org/10.48550/arXiv.1909.01285>
- Zhang, P., Li, C., & Wang, C. (2021). VisCode: Embedding Information in Visualization Images using Encoder-Decoder Network. *IEEE Transactions on Visualization and Computer Graphics*, 27(2), 326–336. <https://doi.org/10.1109/TVCG.2020.3030343>
- Zhang, R., Isola, P., Efros, A. A., Shechtman, E., & Wang, O. (2018). *The Unreasonable Effectiveness of Deep Features as a Perceptual Metric* (arXiv:1801.03924). arXiv. <https://doi.org/10.48550/arXiv.1801.03924>
- Zhang, Y., Ni, J., Su, W., & Liao, X. (2023). A Novel Deep Video Watermarking Framework with Enhanced Robustness to H.264/AVC Compression. *Proceedings of the 31st ACM International Conference on Multimedia*, 8095–8104. <https://doi.org/10.1145/3581783.3612270>
- Zhu, J., Kaplan, R., Johnson, J., & Fei-Fei, L. (2018). HiDDeN: Hiding Data With Deep Networks. *Computer Vision – ECCV 2018: 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XV*, 682–697. [https://doi.org/10.1007/978-3-030-01267-0\\_40](https://doi.org/10.1007/978-3-030-01267-0_40)